FOURTH EDITION

Applied Statistics and the SAS Programming Language

Ronald P. Cody
ROBERT WOOD JOHNSON MEDICAL SCHOOL

Jeffrey K. Smith
RUTGERS UNIVERSITY

PRENTICE HALL, Upper Saddle River, New Jersey 07458

FOURTH EDITION

Applied Statistics and the SAS Programming Language

Ronald P. Cody
ROBERT WOOD JOHNSON MEDICAL SCHOOL

Jeffrey K. Smith
RUTGERS UNIVERSITY

PRENTICE HALL, Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Cody, Ronald P.
Applied statistics and the SAS programming language / Ronald P. Cody, Jeffrey K. Smith – 4th ed.
p. cm.
Includes bibliographical references (p. –) and index.
ISBN 0-13-743642-4 (pbk.)
1. SAS (Computer file) 2. Mathematical statistics–Data processing. I. Smith, Jeffrey K. II. Title.
QA276.4.C53 1997
519.5'0285'5369–dc21 97-1737
CIP

Acquisition editor: Ann Heath
Editorial assistant: Mindy Ince
Editorial director: Tim Bozik
Editor-in-chief: Jerome Grant
Assistant vice president of production and manufacturing: David W. Riccardi
Editorial/production supervision: Nicholas Romanelli
Managing editor: Linda Mihatov Behrens
Executive managing editor: Kathleen Schiaparelli
Manufacturing buyer: Alan Fischer
Manufacturing manager: Trudy Pisciotti
Marketing manager: Melody Marcus
Marketing assistant: Jennifer Pan
Creative director: Paula Maylahn
Cover designer: Jayne Conte

Cover illustration: From the manuscripts of Leonardo DaVinci; published by Charles Raviasson-Mollien, 6 vols., Paris, 1881–91.

Printed in the United States of America

ISBN: 0-13-743642-4

Prentice-Hall, International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To our parents,

Ralph and Bettie Smith

and

Philip and Margaret Cody

Contents

Applied Statistics and SAS Software

SAS Programming

Preface to the Fourth Edition

When we began creating this fourth edition, several facts were clear: First, SAS software continues to evolve and improve. Second, our programming techniques have also improved. Third, several statistical techniques (such as logistic regression) have become popular and required coverage in this edition.

We have met many readers of earlier editions at meetings and conferences and were delighted to hear good things and constructive criticisms of the book. These we have taken to heart and attempted to improve old material and add relevant new topics. This fourth edition is the result of such reader reaction.

Most researchers are inherently more interested in the substance of their research endeavors than in statistical analyses or computer programming. Yet, conducting such analyses is an integral link in the research chain (all too frequently, the weak link). This condition is particularly true when there is no resource for the applied researcher to refer to for assistance in running computer programs for statistical analyses. *Applied Statistics and the SAS Programming Language* is intended to provide the applied researcher with the capacity to perform statistical analyses with SAS software without wading through pages of technical documentation.

The reader is provided with the necessary SAS statements to run programs for most commonly used statistics, explanations of the computer output, interpretations of results, and examples of how to construct tables and write up results for reports and journal articles. Examples have been selected from business, medicine, education, psychology, and other disciplines.

SAS software is a combination of a statistical package, a data-base management system, and a high-level programming language. Like SPSS, BMDP, Systat, and other statistical packages, SAS software can be used to describe a collection of data and produce a variety of statistical analyses. However, SAS software is much more than just a statistical package. Many companies and educational institutions use SAS software as a high-level data-management system and programming language. It can be used to organize and transform data and to create reports of all kinds. Also, depending on which portions of the SAS system you have installed in your computer (and what type of computer system you are running), you may be using the SAS system for interactive data entry or an on-line system for order entry or retrieval.

This book concentrates on the use of the SAS system for the statistical analysis of data and the programming capabilities of SAS software most often used in educational and research applications.

The SAS system is a collection of products, available from the SAS Institute in Cary, North Carolina. The major products available from the SAS Institute are:

Base SAS ®	The main SAS module, which provides some data manipulation and programming capability and some elementary descriptive statistics
SAS/STAT ®	The SAS product that includes all the statistical programs except the elementary ones supplied with the base package
SAS/GRAPH ®	A package that provides high-quality graphs and maps. Note that "line graphics" (the graphs and charts that are produced by normal character plots) are available in the base and SAS/STAT packages. SAS/GRAPH adds the ability to produce high-quality camera-ready graphs, maps, and charts.
SAS/FSP ®	These initials stand for the Full Screen Product. This package allows you to search, modify, or delete records directly from a SAS data file. It also provides for data entry with sophisticated data checking capabilities. Procedures available with FSP are FSBROWSE, FSEDSIT, FSPRINT, FSLIST, and FSLETTER.
SAS/AF ®	AF stands for the SAS Applications Facility. This product is used by data processing professionals to create "turn key" or menu systems for their users. It is also used to create instructional modules relating to the SAS system.
SAS/ETS ®	The Econometric and Time Series package. This package contains specialized programs for the analysis of time series and econometric data.
SAS/OR ®	A series of operations research programs.
SAS/QC ®	A series of programs for quality control.
SAS/IML ®	The Interactive Matrix Language module. The facilities of IML used to be included in PROC MATRIX in the version 5 releases. This very specialized package allows for convenient matrix manipulation for the advanced statistician.

SAS software now runs on a variety of computers, from personal computers to large multimillion dollar mainframes. The original version of the SAS system was written as a combination of PL/1 and IBM assembly language. Today, SAS software runs under Windows and Windows 95 on IBM compatible minicomputers, on most Macintosh computers, under UNIX on a large number of minicomputers and workstations, on IBM computers under a variety of operation systems, on Digital Equipment VAX computers, and others too numerous to mention. The major reason for the ability of SAS software to run on such a variety of machines is that all SAS software was rewritten in C and designed so that most of the code was system-independent. The conversion of the entire system to the C programming language was one of the largest (and most successful) programming projects ever undertaken. To migrate the SAS system to another computer or operating system, only a small system-dependent portion of code needs to be rewritten. The result is that new versions of SAS software

are made available for all computers very quickly, and the versions of SAS systems from one computer to another are much alike.

Learning to program is a skill that is difficult to teach well. While you will find our examples "reader-friendly" and their logic easy to follow, learning to write your own programs is another matter. Only through lots of practice will your programming skills develop. So, when you finish a chapter, please spend the time doing as many problems as you can. We wish you happy programming.

We express our gratitude to two colleagues who reviewed the manuscript and made many comments beneficial to this revision of the text. Our thanks, therefore, to Sylvia Brown and Robert Hamer, at the Robert Wood Johnson Medical School. Our sincere thanks also to Ann Heath, acquisitions editor for statistics at Prentice Hall, for her encouragement and support, and to Nicholas Romanelli for his superb editing, patience, and good cheer.

RON CODY
JEFFREY SMITH

CHAPTER

1

A SAS Tutorial

A. Introduction

For the novice, engaging in statistical analysis of data can seem as appealing as going to the dentist. If that pretty much describes your situation, perhaps you can take comfort in the fact that this is the fourth edition of this book—meaning that the first three editions sold pretty well, and this time we may get it right. Our purpose for this tutorial is to get you started using SAS software. The key objective is to get one program to run successfully. If you can do that, you can branch out a little bit at a time. Your expertise will grow.

The SAS System is a combination of programs originally designed to perform statistical analysis of data. Other programs you may have heard of are SPSS, BMDP, or SYSTAT. If you look at personal computer magazines, you might run across other programs, primarily designed to run on personal computers. Since its inception, the SAS system has grown to where it can perform a fairly impressive array of nonstatistical functions. We'll get into a little of that in later chapters. For now, we want to learn the most basic rudiments of the SAS system. If you skipped over it, the Preface to the fourth edition contains some history of SAS software development and a more complete overview of the capabilities of SAS software.

To begin, SAS software runs on a wide variety of computers and operating systems (computer people call these platforms), and we don't know which one you have. You may have an IBM compatible personal computer running Windows or, perhaps, Windows 95. You may have a Macintosh computer, or you may be connected to a network or a mainframe computer by way of a modem or network

connection. You may only have a sophisticated VCR, which you think is a computer. If you are unsure of what platform you are using or what version of SAS software you are using, ask someone. As a matter of fact, right now would be a good time to invite the best computer person you know to lunch. Have that person arrive at your office about an hour before lunch so you can go over some basic elements of your system. You need to find out what is necessary on your computer to get the SAS system running. What we can teach you here is how to use the SAS system, and how to adapt to your computer system.

If you are running on a mainframe, you may well be submitting what are called "batch" jobs. When you run batch jobs, you send your program (across a phone line or a network from your personal computer or terminal) to the computer. The computer runs your program and holds it until you ask for it or prints out the results on a high-speed printer. You may even need to learn some Job Control Language (which you have to get from your local computer folks), and then you can proceed.

If you are running on a personal computer, or running in what is called interactive mode on a minicomputer or mainframe, then you need to learn how to use the SAS Display Manager. The look and feel of SAS once you are in the Display Manager is pretty much the same whatever platform you are using. If you are undaunted, take a deep breath and plunge into the real content in the next section. If you are already daunted, take a minute and get that lunch scheduled, then come back to this.

B. *Computing with SAS Software: An Illustrative Example*

SAS programs communicate with the computer by SAS "statements." There are several kinds of SAS statements, but they share a common feature—they end in a semicolon. A semicolon in a SAS program is like a period in English. Probably the most common error found in SAS programs is the omission of the semicolon. This omission causes the computer to read two statements as a run-on statement and invariably fouls things up.

SAS programs are comprised of SAS statements. Some of these statements provide information to the system, such as how many lines to print on a page and what title to print at the top of the page. Other statements act together to create SAS data sets, while other SAS statements act together to run predefined statistical or other routines. Groups of SAS statements that define your data and create a SAS data set are called a DATA step; SAS statements that request predefined routines are called a PROC (pronounced "prock") step. DATA steps tell SAS programs about your data. They are used to indicate where the variables are on data lines, what you want to call the variables, how to create new variables from existing variables, and several other functions we mention later. PROC (short for PROCEDURE) steps indicate what kind of statistical analyses to perform and provide specifications for those analyses. Let's look at an example. Consider this simple data set:

SUBJECT NUMBER	GENDER (M or F)	EXAM 1	EXAM 2	HOMEWORK GRADE
10	M	80	84	A
7	M	85	89	A
4	F	90	86	B
20	M	82	85	B
25	F	94	94	A
14	F	88	84	C

We have five variables (SUBJECT NUMBER, GENDER, EXAM 1, EXAM 2, and HOMEWORK GRADE) collected for each of six subjects. The unit of analysis, people for this example, is called an observation in SAS terminology. SAS software uses the term "variable" to represent each piece of information we collect for each observation. Before we can write our SAS program, we need to assign a variable name to each variable. We do this so that we can distinguish one variable from another when doing computations or requesting statistics. SAS variable names must conform to a few simple rules: They must start with a letter, be not more than eight characters (letters or numerals) in length, and cannot contain blanks or certain special characters such as commas, semicolons, etc. (The underscore character (_) is a valid character for SAS variable names and can be used to make variable names more readable.) Therefore, our column headings of "SUBJECT NUMBER," or "EXAM 1" are not valid SAS variable names. Logical SAS variable names for this collection of data would be:

```
SUBJECT     GENDER     EXAM1     EXAM2     HWGRADE
```

It is prudent to pick variable names that help you remember which name goes with which variable. We could have named our five variables VAR1, VAR2, VAR3, VAR4, and VAR5, but we would then have to remember that VAR1 stands for "SUBJECT NUMBER," and so forth.

To begin, let's say we are interested only in getting the class means for the two exams. In reality it's hardly worth using a computer to add up six numbers, but it does provide a nice example. In order to do this, we could write the following SAS program:

```
DATA TEST;  ①
   INPUT SUBJECT 1-2 GENDER $ 4 EXAM1 6-8 EXAM2 10-12  ②
   HWGRADE $ 14;
DATALINES;  ③
10 M  80  84 A
 7 M  85  89 A
 4 F  90  86 B
20 M  82  85 B
25 F  94  94 A
14 F  88  84 C
;
PROC MEANS DATA=TEST; ④
RUN; ⑤
```

The first four lines make up the DATA step. In this example, the DATA step begins with the word DATA and ends with the word DATALINES. Older versions of SAS software used the term CARDS instead of DATALINES. Either term is still valid. (If you don't know what a computer card is, ask an old person.) Line ① tells the program that we want to create a SAS data set called TEST. The next two lines ② show an INPUT statement which gives the program two pieces of information: what to call the variables and where to find them on the data line. Notice that this single SAS statement occupies two lines. The SAS system understands this is a single SAS statement because there is a single semicolon at the end of it. The first variable is SUBJECT and can be found in columns 1 and 2 of the data line. The second variable is GENDER and can be found in column 4. The dollar sign after GENDER means that GENDER is a character (alphanumeric) variable, that is, a variable that can have letters or numbers as data values. (More on this later.) EXAM1 is in columns 6–8, etc. The DATALINES statement ③ says that the DATA statements are done and the next thing the program should look for are the data themselves. The next six lines are the actual data. In this example, we are including the data lines directly in the program. Later on in this book, we will show you how to read data from external files.

Great latitude is possible when putting together the data lines. Using a few rules will make life much simpler for you. These are not laws; they are just suggestions. First, put each new observation on a new line. Having more than one line per observation is often necessary (and no problem), but don't put two observations on one line (at least for now). Second, line up your variables. Don't put EXAM1 in columns 6–8 on one line and in columns 9–11 on the next. SAS software can actually handle some degree of sloppiness here, but sooner or later it'll cost you. Third, right-justify your numeric data values. If you have AGE as a variable, record the data as follows:

Correct	Problematic
87	87
42	42
9	9
26	26
4	4
Right-justified	Left-justified

Again, SAS software doesn't care whether you right-justify numeric data or not, but other statistical programs will, and right justification is standard. Fourth, use character variables sparingly. Take HWGRADE, for example. We have HWGRADE recorded as a character value. But we could have recorded it as 0–4 (0 = F, 1 = D, etc.). As it stands, we cannot compute a mean grade. Had we coded HWGRADE numerically, we could get an average grade. Enough on how to code data for now.

Back to our example. A SAS program knows that the data lines are completed when it finds a SAS statement or a single semicolon. When you include your data lines in the program, as in this example, we recommend placing a single semicolon on the line directly below your last line of data. The next SAS statement ④ is a PROC statement. PROC says "Run a procedure" to the program. We specify which

procedure right after the word PROC. Here we are running a procedure called MEANS. Following the procedure name (MEANS), we place the option DATA= and specify that this procedure should compute statistics on the data set called TEST. In this example, we could omit the option DATA=TEST, and the procedure would operate on the most recently created SAS data set, in this case, TEST. We recommend that you include the DATA= option on every procedure since, in more advanced SAS programs, you can have procedures that create data sets as well as many data sets "floating around." By including the DATA= option, you can always be sure your procedure is operating on the correct data set.

The MEANS procedure calculates the mean for any variables you specify. The RUN statement ⑤ is necessary only when SAS programs are run under the Display Manager. The RUN statement tells SAS that there are no more statements for the preceding procedure and to go ahead and do the calculations. If we have several PROCs in a row, we only need a single RUN statement at the end of the program. However, as a stylistic standard, we prefer to end every procedure with a RUN statement and to separate procedures by a blank line to make the programs more readable.

When this program is executed, it produces something called the SAS LOG and the SAS OUTPUT. The SAS LOG is an annotated copy of your original program (without the data listed). It's a lot like a phone book: Usually it's pretty boring, but sometimes you need it. Any SAS error messages will be found there, along with information about the data set that was created. The SAS LOG for this program is shown below:

```
NOTE: Copyright (c) 1989-1995 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 6.11  TS020
      Licensed to RON CODY/ROBERT WOOD JOHNSON MEDICAL SCHOOL, Site XXX.

NOTE: Release 6.11 of the SAS(R) System for Windows(R).

NOTE: AUTOEXEC processing completed.

1
2          DATA TEST;
3             INPUT SUBJECT 1-2 GENDER $ 4 EXAM1 6-8 EXAM2 10-12
4             HWGRADE $ 14;
5          DATALINES;

NOTE: The data set WORK.TEST has 6 observations and 5 variables.
NOTE: The DATA statement used 0.27 second.


12         ;
13         PROC MEANS DATA=TEST;
14         RUN;

NOTE: The PROCEDURE MEANS used 0.17 second.
```

The more important part of the output is found in the OUTPUT window if you are using the Display Manager. It contains the results of the computations and procedures requested by our PROC statements. This portion of the output from the program above is shown next:

Variable	N	Mean	Std Dev	Minimum	Maximum
SUBJECT	6	13.3333333	7.9916623	4.0000000	25.0000000
EXAM1	6	86.5000000	5.2057660	80.0000000	94.0000000
EXAM2	6	87.0000000	3.8987177	84.0000000	94.0000000

If you don't specify which variables you want, SAS software will calculate the mean and several other statistics for every numeric variable in the data set. Our program calculated means for SUBJECT, EXAM1, and EXAM2. Since SUBJECT is just an arbitrary ID number assigned to each student, we aren't really interested in its mean. We can avoid getting it (and using up extra CPU cycles) by adding a new statement under PROC MEANS:

```
PROC MEANS DATA=TEST;
   VAR EXAM1 EXAM2;    ⑥
RUN;
```

The indentation used is only a visual aid. The VAR statement ⑥ specifies on which variables to run PROC MEANS. PROC MEANS not only gives you means, it gives you the number of observations used to compute the mean, the standard deviation, the minimum score found, and the maximum score found. PROC MEANS can compute many other statistics such as variance and standard error. You can specify just which pieces you want in the PROC MEANS statement. For example:

```
PROC MEANS DATA=TEST N MEAN STD STDERR MAXDEC=1;
   VAR EXAM1 EXAM2;
RUN;
```

will get you only the number of observations with no missing values (N), mean (MEAN), standard deviation (STD), and standard error (STDERR) for the variables EXAM1 and EXAM2. In addition, the statistics will be rounded to one decimal place (because of the MAXDEC=1 option). Chapter 2 describes most of the commonly requested options used with PROC MEANS.

C. *Enhancing the Program*

The program as it is currently written provides some useful information but, with a little more work, we can put some "bells and whistles" on it. The bells and whistles version below adds the following features: It computes a final grade, which we will let be the average of the two exam scores; it assigns a letter grade based on that final score; it lists the students in student number order, showing their exam scores, their final grades and homework grades; it computes the class average for the exams and final grade and a frequency count for gender and homework grade; finally, it gets you a cup of coffee and tells you that you are a fine individual.

```
DATA EXAMPLE;  ①
   INPUT SUBJECT GENDER $ EXAM1 EXAM2 ②
   HWGRADE $;
FINAL = (EXAM1 + EXAM2) / 2; ③
IF FINAL GE 0 AND FINAL LT 65 THEN GRADE='F';  ④
⑤ ELSE IF FINAL GE 65 AND FINAL LT 75 THEN GRADE='C';
   ELSE IF FINAL GE 75 AND FINAL LT 85 THEN GRADE='B';
   ELSE IF FINAL GE 85 THEN GRADE='A';
DATALINES;  ⑥
10 M  80  84 A
 7 M  85  89 A
 4 F  90  86 B
20 M  82  85 B
25 F  94  94 A
14 F  88  84 C
;
PROC SORT DATA=EXAMPLE;  ⑦
   BY SUBJECT;  ⑧
RUN;  ⑨

PROC PRINT DATA=EXAMPLE;  ⑩
   TITLE 'Roster in Student Number Order';
   ID SUBJECT;
   VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;

PROC MEANS DATA=EXAMPLE N MEAN STD STDERR MAXDEC=1; ⑪
   TITLE 'Descriptive Statistics';
   VAR EXAM1 EXAM2 FINAL;
RUN;

PROC FREQ DATA=EXAMPLE; ⑫
   TABLES GENDER HWGRADE GRADE;
RUN;
```

As before, the first four lines constitute our DATA step. Line ① is an instruction for the program to create a data set whose data set name is "EXAMPLE." (Remember that data set names follow the same conventions as variable names.) Line ② is an INPUT statement which is different from the one in the previous example.

We could have used the same INPUT statement as in the previous example but wanted the opportunity to show you another way that SAS programs can read data. Notice that there are no column numbers following the variable names.

This form of an INPUT statement is called list input. To use this form of INPUT, the data values must be separated by one or more blanks (or other separators which computer people call delimiters). If you use one of the other possible delimiters, you need to modify the program accordingly (see Chapter 12, Section C). The order of the variable names in the list corresponds to the order of the values in the line of data. In this example, the INPUT statement tells the program that the first variable in each line of data represents SUBJECT values, the next variable is GENDER, the third EXAM1, and so forth. If your data conform to this "space-between-each-variable" format, then you don't have to specify column numbers for each variable listed in the INPUT statement. You may want to anyway, but it isn't necessary. (You still have to follow character variable names with a dollar sign.) If you are going to use "list input," then every variable on your data lines must be listed. Also, since the order of the data values is used to associate values with variables, we have to make special provisions for missing values. Suppose that subject number 10 (the first subject in our example) did not take the first exam. If we listed the data like this:

```
10 M    84 A
```

with the EXAM1 score missing, the 84 would be read as the EXAM1 score, the program would read the letter "A" as a value for EXAM2 (which would cause an error since the program was expecting a number), and, worst of all, the program would look on the next line for a value of homework grade. You would get an error message in the SAS LOG telling you that you had an invalid value for EXAM2 and that SAS went to a new line when the INPUT statement reached past the end of a line. You wouldn't understand these error messages and might kick your dog.

To hold the place of a missing value when using a list INPUT statement, use a period to represent the missing value. The period will be interpreted as a missing value by the program and will keep the order of the data values intact. When we specify columns as in the first example, we can use blanks as missing values. Using periods as missing values when we have specified columns in our INPUT statement is also OK, but not recommended. The correct way to represent this line of data, with the EXAM1 score missing, is:

```
10 M . 84 A
```

Since list input requires one or more blanks between data values, we need at least one blank before and after the period. We may choose to add extra spaces in our data to allow the data values to line up in columns.

Line ③ is a statement assigning the average of EXAM1 and EXAM2 to a variable called FINAL. The variable name "FINAL" must conform to the same naming conventions as the other variable names in the INPUT statement. In this example, FINAL is calculated by adding together the two exam scores and dividing by 2. Notice that we indicate addition with a + sign and division by a / sign. We need the parentheses because, just the same as in handwritten algebraic expressions, SAS

computations are performed according to a hierarchy. Multiplication and division are performed before addition and subtraction. Thus, had we written:

```
FINAL = EXAM1 + EXAM2 / 2;
```

the FINAL grade would have been the sum of the EXAM1 score and half of EXAM2. The use of parentheses tells the program to add the two exam scores first, and then divide by 2. To indicate multiplication, we use an asterisk (*); to indicate subtraction, we use a - sign. Exponentiation, which is performed before multiplication or division, is indicated by two asterisks. As an example, to compute A times the square of B we write:

```
X = A * B**2;
```

The variable FINAL was computed from the values of EXAM1 and EXAM2. That does not, in any way, make it different from the variables whose values were read in from the raw data. When the DATA step is finished, the SAS procedures that follow will not treat variables such as FINAL any differently from variables such as EXAM1 and EXAM2.

The IF statement ④ and the ELSE IF statements ⑤ are logical statements that are used to compute a letter grade. They are fairly easy to understand. When the condition specified by the IF statement is true, the instructions following the word THEN are executed. The logical comparison operators used in this example are GE (greater than or equal to) and LT (less than). So, if a FINAL score is greater than or equal to 0, and less then 65, a letter grade of 'F' is assigned. The ELSE statements are only executed if a previous IF statement is not true. For example, if a FINAL grade is 73, the first IF statement ⑤ is not true, so the ELSE IF statement ⑥ is tested. Since this statement is true, a GRADE of 'C' is assigned, and all the following ELSE IF statements are skipped.

Other logical operators and their equivalent symbolic form are shown in the table below:

Expression	Symbol	Meaning
EQ	=	Equal
LT	<	Less than
LE	< =	Less than or equal
GT	>	Greater than
GE	> =	Greater than or equal
NE	^=	Not equal
NOT	^	Negation

NOTE: The symbols for NOT and NE may vary, depending on your system.

The "DATALINES" statement ⑥ indicates that the DATA step is complete and that the following lines contain data.

Notice that each SAS statement ends with a semicolon. As mentioned before, the semicolon is the logical end of a SAS statement. We could have written the first four lines like this:

```
DATA EXAMPLE; INPUT SUBJECT GENDER $
EXAM1 EXAM2 HWGRADE $; FINAL =
(EXAM1 + EXAM2) / 2;
```

The program would still run correctly. Using a semicolon as a statement delimiter is convenient since we can write long SAS statements on several lines and simply put a semicolon at the end of the statement. However, if you omit a semicolon at the end of a SAS statement, the program will attempt to read the next statement as part of previous statement, causing an error. This may not only cause your program to die, it may result in a bizarre error message emanating from the SAS system. Omission of one or more semicolons is the most common programming error for novice SAS programmers. Remember to watch those semicolons! Notice also that the data lines, since they are not SAS statements, do not end with semicolons. In fact, under most usual circumstances, data are not allowed to contain semicolons.

Following the DATALINES statement are our data lines. Remember that if you have data that have been placed in preassigned columns with no spaces between the data values, you must use the form of the INPUT shown earlier, with column specifications after each variable name. This form of data is discussed further in Chapter 12. We have used a RUN statement to end every procedure. Each RUN statement tells the system that we are finished with a section of the program and to do the computations just concluded. Remember, when using the Display Manager, only the last RUN statement is absolutely necessary; the others are really only a matter of programming style.

D. *SAS Procedures*

Immediately following the data is a series of PROCs. They perform various functions and computations on SAS data sets. Since we want a list of subjects and scores in subject order, we first include a SORT PROCEDURE ⑦, ⑧, and ⑨. Line ⑦ indicates that we plan to sort our data set; line ⑧ indicates that the sorting will be by SUBJECT number. Sorting can be multilevel if desired. For example, if we want separate lists of male and female students in subject number order, we write:

```
PROC SORT DATA=EXAMPLE;
  BY GENDER SUBJECT;
RUN;
```

This multilevel sort indicates that we should first sort by GENDER (F's followed by M's—character variables are sorted alphabetically), then in SUBJECT order within GENDER.

The PRINT procedure ⑩ requests a listing of our data (which is now in SUBJECT order). The PRINT procedure is used to list the data values in a SAS data set. We have followed our PROC PRINT statement with three statements that supply information to the procedure. These are the TITLE, ID, and VAR statements. As with many SAS procedures, the supplementary statements following a PROC can be placed in any order. Thus:

```
PROC PRINT DATA=EXAMPLE;
   ID SUBJECT;
   TITLE 'Roster in Student Number Order';
   VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;
```

is equivalent to

```
PROC PRINT DATA=EXAMPLE;
  TITLE 'Roster in Student Number Order';
  ID SUBJECT;
  VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;
```

SAS programs recognize the keywords TITLE, ID, and VAR and interpret what follows in the proper context. Notice that each statement ends with its own semicolon. The words following TITLE are placed in single (or double) quotes and will be printed across the top of each of the SAS output pages. The ID variable, SUBJECT in this case, will cause the program to print the variable SUBJECT in the first column of the report, omitting the column labeled OBS (observation number) which the program will print when an ID variable is absent. The variables following the keyword VAR indicate which variables, besides the ID variable, we want in our report. The order of these variables in the list also controls the order in which they appear in the report.

The MEANS procedure ⑪ is the same as the one we used previously. Finally, the FREQ procedure ⑫ (you're right: pronounced "PROC FREAK") requests a frequency count for the variables GENDER, HWGRADE, and GRADE. That is, what is the number of Males and Females, the number of A's, B's, etc., as well as the percentages of each category. PROC FREQ will compute frequencies for the variables listed on the TABLES statement. The reason that SAS uses the keyword TABLES instead of VAR for this list of variables is that PROC FREQ can also produce n-way tables (such as 2×3 tables).

Output from the complete program is shown below:

```
Roster in Student Number Order                                    1
                                        13:15 Wednesday, July 31, 1996

SUBJECT     EXAM1     EXAM2     FINAL     HWGRADE     GRADE

    4          90        86      88.0         B          A
    7          85        89      87.0         A          A
   10          80        84      82.0         A          B
   14          88        84      86.0         C          A
   20          82        85      83.5         B          B
   25          94        94      94.0         A          A

Descriptive Statistics          13:15 Wednesday, July 31, 1996   2

Variable  N           Mean        Std Dev      Std Error
-----------------------------------------------------------
EXAM1     6           86.5            5.2            2.1
EXAM2     6           87.0            3.9            1.6
FINAL     6           86.8            4.2            1.7
-----------------------------------------------------------

Descriptive Statistics          13:15 Wednesday, July 31, 1996   3

                                    Cumulative  Cumulative
GENDER    Frequency    Percent     Frequency    Percent
-----------------------------------------------------------
F                3       50.0             3       50.0
M                3       50.0             6      100.0

                                    Cumulative  Cumulative
HWGRADE   Frequency    Percent     Frequency    Percent
-----------------------------------------------------------
A                3       50.0             3       50.0
B                2       33.3             5       83.3
C                1       16.7             6      100.0

                                    Cumulative  Cumulative
GRADE     Frequency    Percent     Frequency    Percent
-----------------------------------------------------------
A                4       66.7             4       66.7
B                2       33.3             6      100.0
```

The first part of the output (the page number is shown at the extreme right of each page) is the result of the PROC PRINT on the sorted data set. Each column is labeled with the variable name. Because we used an ID statement with SUBJECT as the ID variable, the left-most column shows the SUBJECT number instead of the default OBS column, which would have been printed if we did not have an ID variable.

Page 2 of the output lists each of the variables listed on the VAR statement and produces the requested statistics (N, mean, standard deviation, and standard error) all to the tenths place (because of the MACDEX=1 option).

Page 3 is the result of the PROC FREQ request. Notice that the title "Descriptive Statistics" is still printed at the top of each page. Titles remain in effect until the end of the current SAS session or if you change it to another title line. This portion of the listing gives you frequencies (the number of observations with particular values) as well as percentages. The two columns labeled "Cumulative Frequency" and "Cumulative Percent" are not really useful in this example. In other cases, where a variable represents an ordinal quantity, the cumulative statistics may be more useful.

E. *Overview of the SAS DATA Step*

Let's spend a moment examining what happens when we execute a SAS program. This discussion is a bit technical and can be skipped, but an understanding of how SAS software works will help when you are doing more advanced programming. When the DATA statement is executed, SAS software allocates a portion of a disk and names the data set "EXAMPLE," our choice for a data set name. Before the INPUT statement is executed, each of the character and numeric variables is assigned a missing value. Next, the INPUT statement reads the first line of data and substitutes the actual data values for the missing values. These data values are not yet written to our SAS data set EXAMPLE but to a place called the Program Data Vector (PDV). This is simply a "holding" area where data values are stored before they are transferred to the SAS data set. The computation of the final grade comes next ③, and the result of this computation is added to the PDV. Depending on the value of the final grade, a letter grade is assigned by the series of IF and ELSE IF statements. The DATALINES line triggers the end of the DATA step, and the values in the PDV are transferred to the SAS data set. The program then returns control back to the INPUT statement to read the next line of data, compute a final grade, and write the next observation to the SAS data set. This reading, processing, and writing cycle continues until no more observations remain. Wasn't that interesting?

F. *Syntax of SAS Procedures*

As we have seen above, SAS procedures can have options. Also, procedures often have statements, like the VAR statement above, which supply information to the procedure. Finally, statements can also have options. We will show you the general syntax of SAS procedures and then illustrate it with some examples. The syntax for all SAS procedures is:

```
PROC PROCNAME options;
   STATEMENTS / statement options;
            .
            .
            .
   STATEMENTS / statement options;
RUN;
```

First, all procedures start with the word PROC followed by the procedure name. If there are any procedure options, they are placed, in any order, between the procedure name and the semicolon, separated by spaces. If we refer to a SAS manual, under PROC MEANS we will see a list of options to be used with the procedure. As mentioned, N, MEAN, STD, STDERR, and MAXDEC= are some of the available options. A valid PROC MEANS request for statistics from a data set called EXAMPLE, with options for N, MEAN,and MAXDEC would be:

```
PROC MEANS DATA=EXAMPLE N MEAN MAXDEC=1;
RUN;
```

Next, most procedures need statements to supply more information about which type of analysis to perform. An example would be the VAR statement used with PROC MEANS. Statements follow the procedure, in any order. They each end with a semicolon. So, to run the PROC MEANS statement above, on the variables EXAM1 and EXAM2, and to supply a title, we would enter:

```
PROC MEANS DATA=EXAMPLE N MEAN STD MAXDEC=1;
   TITLE 'Descriptive Statistics on Exam Scores';
   VAR EXAM1 EXAM2;
RUN;
```

The order of the TITLE and VAR statements can be interchanged with no change in the results. Finally, some procedure statements also have options. Statement options are placed between the statement keyword and the semicolon and separated from the statement by a slash. To illustrate, we need to choose a procedure other than PROC MEANS. Let's use PROC FREQ as an example. As we saw, PROC FREQ will usually have one or more TABLES statements following it. There are TABLES options that control which statistics can be placed in a table. For example, if we do not want the cumulative statistics printed, the statement option NOCUM is used. Since this is a statement option, it is placed between the TABLES statement and the semicolon, separated by a slash. The PROC FREQ request in the earlier example, modified to remove the cumulative statistics, would be:

```
PROC FREQ DATA=EXAMPLE;
   TABLES GENDER HWGRADE GRADE/ NOCUM;
RUN;
```

To demonstrate a procedure with procedure options and statement options, we use the ORDER= option with PROC FREQ. This useful option controls the order that the values can be arranged in our frequency table. One option is

ORDER=FREQ, which enables the frequency table to be arranged in frequency order, from the highest frequency to the lowest. So, to request frequencies in descending order of frequency and to omit cumulative statistics from the output, we write our PROC FREQ statements as follows:

```
PROC FREQ DATA=EXAMPLE ORDER=FREQ;
   TABLES GENDER HWGRADE GRADE/ NOCUM;
RUN;
```

G. Comment Statements

Before concluding this chapter, we introduce one of the most important SAS statements—the comment statement. (Yes, we're not kidding!) A properly commented program indicates that a true professional is at work. A comment, inserted in a program is one or more lines of text that are ignored by the program—they are there only to help the programmer or researcher when he or she reads the program at a later date.

There are two ways to insert comments into a SAS program. One is to write a comment statement. Begin it with an asterisk (*) and end it with a semicolon. There are many possible styles of comments using this method. For example:

```
*Program to Compute Reliability Coefficients
 Ron Cody
 September 18, 1997
 Program Name: FRED stored in directory C:\MYDATA

 Contact Fred Cohen at 555-4567;
```

Notice the convenience of this conclusion. Just enter the * and type as many lines as necessary, ending with the semicolon. Just make sure the comment statement doesn't contain a semicolon. Some programmers get fancy and make pretty boxes for their comments, like this:

```
*-------------------------------------------------------*
 | Program Name: FRED  stored in C:\MYDATA               |
 | Purpose: To compute reliability coefficients          |
 | Contact: Fred Cohen at 555-4567                       |
 | Date: September 18, 1997                              |
 | Programmer: Ron Cody                                  |
*-------------------------------------------------------*;
```

Notice that the entire box is a SAS comment statement since it begins with an asterisk and ends in a semicolon. Notice also that the box literally cries out, "I need a life!"

You may also choose to comment individual lines by resorting to one of the following three ways:

```
QUES = 6 - QUES;  *Transform QUES VAR;
X = LOG(X);  *LOG Transform of X;
```

or

```
*Transform the QUES Variable;
QUES = 6 - QUES;
*Take the LOG of X;
X = LOG(X);
*True professonal at work;
```

or

```
*
*Transform the QUES Variable
*;
QUES = 6 - QUES;
*
*Take the LOG of X
*;
X = LOG(X);
*
*True professonal at work
*;
```

The last example uses more than one asterisk to set off the comment, for visual effect. Note however, that each group of three lines is a single comment statement since it begins with an asterisk and ends with a semicolon.

An alternative commenting method begins a comment with a /* and ends with a */. This form of comment can be embedded within a SAS statement and can include semicolons within the comment itself. It can occur any place a blank can occur. A few examples:

```
/* This is a comment line */
```

or


```
/*-------------------------------------------------*
 | This is a pretty comment box using the slash star |
 | method of commenting.  Notice that it begins with |
 | a slash star and ends with a star slash.          |
 *-------------------------------------------------*/
```

or

```
DATA EXAMPLE;  /* The data statement */
   INPUT SUBJECT GENDER $
         EXAM1 /* EXAM1 is the first exam score */
         EXAM2 /* EXAM2 is the second exam score */
         HWGRADE $;
   FINAL = (EXAM1 + EXAM2)/2;  /* Compute a composite grade */
DATALINES;
```

Let us show you one final, very useful trick using a comment statement, before concluding this chapter. Suppose you have written a program and run several procedures. Now, you return to the program and wish to run additional procedures. You could edit the program, remove the old procedures, and add the new ones. Or, you could "comment them out" by preceding the section with a /* and ending with a */, making the entire section a comment. As an example, our commented program could look like this:

```
DATA MYPROG;
   INPUT X Y Z;
DATALINES;
1 2 3
3 4 5
;
/*******************************************************
PROC PRINT DATA=MYPROG;
   TITLE 'MY TITLE';
   VAR X Y Z;
RUN;
*******************************************************/
PROC CORR DATA=MYPROG;
   VAR X Y Z;
RUN;
```

The print procedure is not executed since it is treated as a comment; the correlation procedure will be run.